

AD-A172 964

A METHOD OF DESCRIBING AND SYNTHESIZING AN ARBITRARY

1/1

PULSE SEQUENCE(U) AIR FORCE INST OF TECH  
WRIGHT-PATTERSON AFB OH S R ALLEN AUG 86

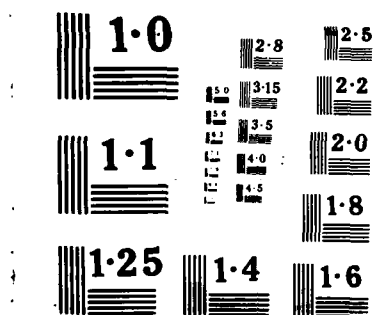
UNCLASSIFIED

AFIT/CI/MR-86-187T

F/G 14/2

NL

END  
DATE  
FILMED  
11-86  
FBI



AD-A172 964

DTIC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 86-187T	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Method of Describing and Synthesizing an Arbitrary Pulse Sequence		5. TYPE OF REPORT & PERIOD COVERED THESIS/DISSERTATION
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Steven Ray Allen		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of Illinois		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 1986
		13. NUMBER OF PAGES 71
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASS
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1		LYNN E. WOLAVER 25/4/86 Dean for Research and Professional Development AFIT/NR
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)  ATTACHED ...		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC  
ELECTE  
OCT 14 1986  
S E D

## **DISCLAIMER NOTICE**

**THIS DOCUMENT IS BEST QUALITY  
PRACTICABLE. THE COPY FURNISHED  
TO DTIC CONTAINED A SIGNIFICANT  
NUMBER OF PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.**

A METHOD OF DESCRIBING AND SYNTHESIZING  
AN ARBITRARY PULSE SEQUENCE

BY

STEVEN RAY ALLEN

B.S., University of South Carolina, 1982

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Electrical Engineering  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 1986

Urbana, Illinois



Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

AUGUST 1986

WE HEREBY RECOMMEND THAT THE THESIS BY

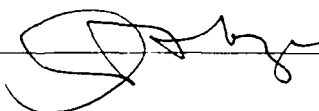
STEVEN RAY ALLEN

ENTITLED A METHOD OF DESCRIBING AND SYNTHESIZING

AN ARBITRARY PULSE SEQUENCE

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE



Director of Thesis Research

Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

# University of Illinois at Urbana-Champaign

## DEPARTMENTAL FORMAT APPROVAL

THIS IS TO CERTIFY THAT THE FORMAT AND QUALITY OF PRESENTATION OF THE  
THESIS SUBMITTED BY STEVEN RAY ALLEN AS ONE OF  
THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE  
IS ACCEPTABLE TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING.  
*(Full Name of Department, Division or Unit)*

August 21, 1986

*Date of Approval*

  
*Departmental Representative*

## ACKNOWLEDGEMENTS

I would like to thank Dr. Gernot Metze for his guidance and support in the production of this thesis. I also appreciate the sponsorship of the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio. Finally, my loving wife, Cathy, has endured the hardships of undergraduate and graduate studies along with me and deserves more thanks than I could give her in my lifetime.



## TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION.....	1
2. PULSE SEQUENCE DEFINITION.....	5
3. TRANSFORM METHODS.....	15
4. HARDWARE AND SOFTWARE METHODS.....	34
5. DESCRIPTION OF AN ARBITRARY SEQUENCE.....	45
6. IMPLEMENTATION OF THE PULSE DESCRIPTION..	55
7. CONCLUSIONS.....	66
BIBLIOGRAPHY.....	69

## CHAPTER 1

### INTRODUCTION

#### 1.0 Thesis outline

This research centers on the problem of how to precisely describe any arbitrarily determined binary pulse sequence. If such a technique is developed, can we then implement this description into some form of device which reproduces the specified signal? In Chapter 1 I examine the impetus for this research and conclude with the problem statement. Chapter 2 contains the concept of an arbitrary pulse sequence and provides the fundamental definitions used throughout the remainder of the thesis. Transform methods using the Fourier and Walsh transforms are examined in Chapter 3. Chapter 4 uses hardware solutions, such as shift register theory, to solve the pulse sequence problem. The problems associated with transform methods and hardwired solutions when applied to the arbitrary sequence problem are highlighted in Chapter 5 with my proposed solution concluding the chapter. Where Chapter 5 contains the specifics of my solution, Chapter 6 proposes various hardware/software implementations of this technique. Concluding remarks and conjectures complete the thesis in Chapter 7.

## 1.1 Background

Binary pulse sequences occur throughout many disciplines. Probably the most recognized application is the field of pulse coded modulation (PCM), which quantizes an analog waveform and produces for each sampled instant a single value represented as a binary word. This word, when viewed with other "words" from previous samples, produces a sequence of binary ones and zeroes which describes the analog input. Pulse coded modulation was made practical in 1936-1937 by A. Reeves, and "Today, PCM is the universal standard for digital transmission of telephone and is used with terrestrial microwave radio, satellites, and fiber optics as well as metallic cable." [1]

Computer buss traffic yields sequences as arbitrary as the programmer determines and the system software supports. This traffic often assumes both random and periodic (recurring) forms of data transmission. If we look at the computer's internal mechanism, we see the output of its combinational and sequential digital circuits describing a binary pulse sequence. In fact, the function column of a truth table simply enumerates a sequence of ones and zeroes that are conditional upon the input variables and the time at which the output is observed.

The triggering of measurement equipment during an evaluation creates another application of pulse sequences. A prime example is a nuclear magnetic resonance (NMR) experiment involving either single or double resonance. Pulse patterns already exist for the Carr-Purcell, WAHUHA, and MREV-8

sequences.[2] These NMR patterns define the critical timing points in the nuclear spectroscopy experiment.

The basis of radar is pulse sequence timing. The time between adjacent pulses determines the unambiguous range of the system. Resolution of two closely spaced targets depends on the width of the radar pulse. Blind speeds and range ambiguities associated with doppler and moving target indicator (MTI) radars are eliminated through the proper blend of pulse spacings. In the realm of electronic warfare (EW), much can be inferred about the operation of a weapon by an analysis of the system's radar pulse train.[3] Incidentally, I was introduced to the concept of the arbitrary pulse sequence and the problems associated with its synthesis while working at the Avionics Laboratory, Electronic Warfare Division, Wright-Patterson Air Force Base, Ohio. As a solution, I created the first generation synthesizer, called the Multi-Agile Pulse Synthesizer; but, without some means of compactly describing a sequence, the MAPS was an inefficient device. It did, however, provide the spark needed to search further into this problem and subsequently spawned this research.

This discussion is by no means a compendium of pulse sequence applications but is instead meant to familiarize the reader with a few applications.

## 1.2 Problem statement

In general, a design engineer has little problem

concocting a design which produces a desired pulse sequence. But let us now force another pulse train upon our unsuspecting engineer! Perhaps the existing device can be altered for the new requirement; perhaps not. Frustrated, our engineer foresees that management really wants a design which is independent of the desired pulse sequence. Herein lies the focus of my thesis: develop a sequence-independent pulse synthesizer; that is, find a design which requires no hardware reconfiguration regardless of the desired pulse train. To solve this problem I will examine what is required to describe an arbitrary pulse sequence. Existing methods will be checked for their applicability, and their shortcomings provide the impetus for my method of description and implementation.

## CHAPTER 2

### PULSE SEQUENCE DEFINITION

#### 2.0 Introduction

Chapter 2 exists so we can share a common basis of terminology and a mutual understanding of the construction of a binary pulse sequence. Many of the terms describing these sequences are borrowed from the field of radar, where these definitions are already well established. Before diving into the material, a fundamental assumption about the construction of a sequence is now stated. I assume only two states are possible in the construction of the pulse sequence. These binary states may be represented in many different fashions. (i.e., on-off keying, polar pulsed signals, and phase shift keying), but there must be only two possible states.

#### 2.1 Single pulse definition

A single pulse is described by unit step functions.[4] By definition, a unit step function,  $u(t)$ , assumes the value of 1 for  $t > 0$  and 0 for  $t < 0$  (Figure 1). Please note that all figures appear at the end of each chapter. The step function is a dimensionless quantity and is generally stated as  $u(t - t_0)$  where  $t_0$  denotes the instant of time where the transition from 0 to 1 occurs (Figure 2). A pulse is created when two step functions with different transition times are

subtracted. Figure 3 illustrates the pulse formed by the function

$$v(t) = u(t - t_0) - u(t - t_1) \quad \text{for } t_1 > t_0$$

The value  $t_1 - t_0$  is called the pulse width (PW); so, the  $PW = t_1 - t_0$  for this example.

## 2.2 Single pulse sequence

Suppose we now create a series of pulses with the  $PW = t_0$  and a spacing between PWs equal to  $T$  where  $T = 2 \times PW$ . Our function now looks like this:

$$v(t) = \sum \{u(t - nT) - u[t - (nT + t_0)]\} \quad \text{for } n = 0 \rightarrow \infty$$

(see Figure 4)

This function represents a square wave of frequency  $1/T$  beginning at the origin. The time interval between leading edges of adjacent pulses is called the pulse repetition interval (PRI); so, the  $PRI = T$  for this example. The pulse repetition frequency (PRF) equals  $1/PRI$ . The value of  $v(t)$  from  $t_0$  to  $T$  represents the OFF time of the sequence and is equal to the  $PRI - PW$ . This will have special significance later on. A note is now appropriate for the terms "ON" and "OFF." For the remainder of this thesis I assume that the asserted (true) state of a signal is the "ON" state and the nonasserted (false) state is the "OFF" state. Further, I assume positive logic is used whenever the logic polarity is important in the discussion. Signals using negative logic appear as negated variables (i.e., low assertive  $x$  is  $\bar{x}$ ).

These assumptions will not affect the results of this research, since I use ON and OFF to label the states.

There is no special requirement that the desired sequence always be a square wave, and in fact the PW and PRI seldom generate a 50% duty cycle. The only requirement imposed on the sequence is that  $T > t_0$  (Figs. 5 & 6). Please note that from now on a "pulse" is described by an ON and an OFF time.

### 2.3 Two pulse sequence

Although the uses for the single valued pulse sequence are nearly infinite, life becomes boring and impractical for many applications because of only one pulse value. Suppose we now wish a sequence of two pulses, each of different PW. Refer to Figure 7 and let

$$\begin{aligned} PW_1 &= t_0 & PW_2 &= t_1 \\ PRI_1 &= T_0 & PRI_2 &= T_1 \end{aligned}$$

The equation for this system is

$$\begin{aligned} v(t) &= [u\{t-n(T_0+T_1)\}-u\{t-(n(T_0+T_1)+t_0)\}] + \\ &\quad [u\{t-((n+1)T_0+nT_1)\}-u\{t-((n+1)T_0+nT_1+t_1)\}] \\ &\quad \text{for } n = 0 \rightarrow \infty \end{aligned}$$

Algebraically, the order of the terms in  $v(t)$  does not matter because of the commutative property of addition and the definition of the unit step function. I have grouped those terms representing the creation of pulse 1 and pulse 2 together and ordered these so that the equation reads from pulse one to



pulse two. There is one square bracket for each pulse in this sequence. Also note that the function  $v(t)$  represents an infinite series, since  $n \rightarrow \infty$ .

#### 2.4 An "N" pulse sequence

A logical extension to Section 2.3 is a pulse sequence containing  $N$  different pulses before the entire burst repeats *ad infinitum*. The  $N$ -pulse equation differs from the 2-pulse equation only in the number of bracketed quantities, and for an  $N$  pulse sequence there exist  $N$  brackets.

This is an opportune time for another assumption regarding these pulse sequences. In the real world we may have truly random pulse sequences, but the arithmetic statement of such a phenomenon, in terms of unit step functions, is infinite in length. Therefore, I assume the pulse sequences discussed are all recursive in nature; that is, there exists a time at which the entire burst repeats into infinity. This assumption produces an unbearably tight restraint in the mathematical formulation of the pulse sequence theory; however, I will show that the assumption creates conceptually simple statements which can be extended to account for the random and pseudorandom occurrences.

#### 2.5 Multiple repetitions of a pulse

The theory of the preceding sections also applies to a

series containing multiple repetitions of a single pulse. One bracket is required for each repetition of the pulse, and each bracket will contain the same PW and PRI information as its predecessor, except for the time shift of each subsequent pulse. Further, if some other pulses in the sequence also contain multiple repetitions, this method must be applied to each of those pulses. In Figure 8 the function  $v(t)$  has 4 brackets with 2 of these being redundant, (i.e., they only provide for duplicate pulses). The equation for  $v(t)$  is shown along with the diagram.

## 2.6 Types of sequences

As you can imagine, the variations for binary pulse sequences are infinite; however, some series occur more frequently than others. Wiley[5] suggests several common PRI categories: constant, jittered, dwell and switch, staggered, sliding, scheduled, periodic variations, and pulse grouping. Admittedly, these classifications were developed for radar, but I submit that these categories also apply to a great many other fields and serve to quantify the many variations in sequences. I feel the addition of random and pseudorandom sequences to these categories extends the list to cover nearly all possibilities.

## 2.7 Summary of assumptions

Before proceeding, let us review the several assumptions

made within this chapter. The sections where the assumptions are explained follow the assumptions.

1. Only binary pulse sequences are examined. (§2.0)
2. ON refers to the asserted (true) state;  
OFF refers to the nonasserted (false) state;  
positive logic is assumed except where noted otherwise.  
(§2.2)
3. A pulse includes both ON and OFF times. (§2.2)
4. All sequences are recurring in nature. (§2.4)

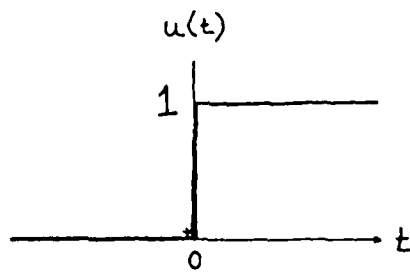


Figure 1 A unit step function for  $t_0 = 0$ .

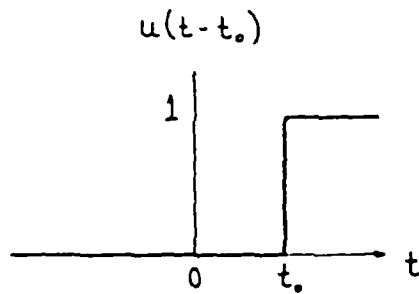


Figure 2 A unit step function for  $t_0 \neq 0$ .

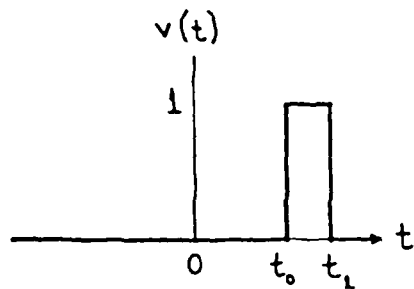


Figure 3 Creation of a pulse from unit step functions.

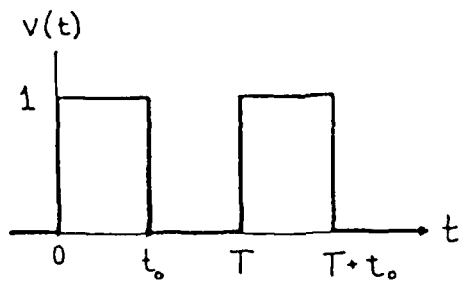


Figure 4 A pulse of  $PW = t_0$  and  $FFI = T$ .

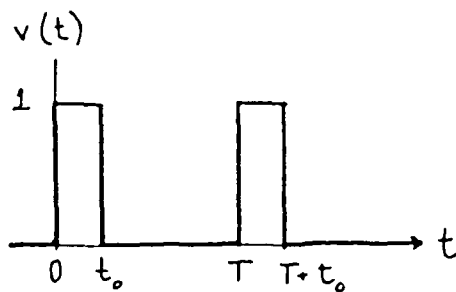


Figure 5 A pulse with less than 50% duty cycle.

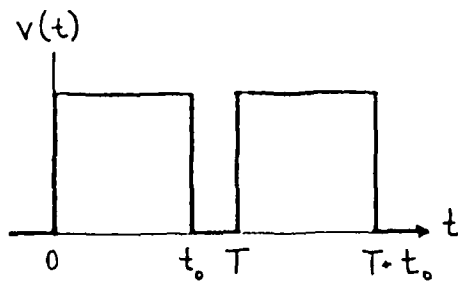


Figure 6 A pulse with greater than 50% duty cycle.

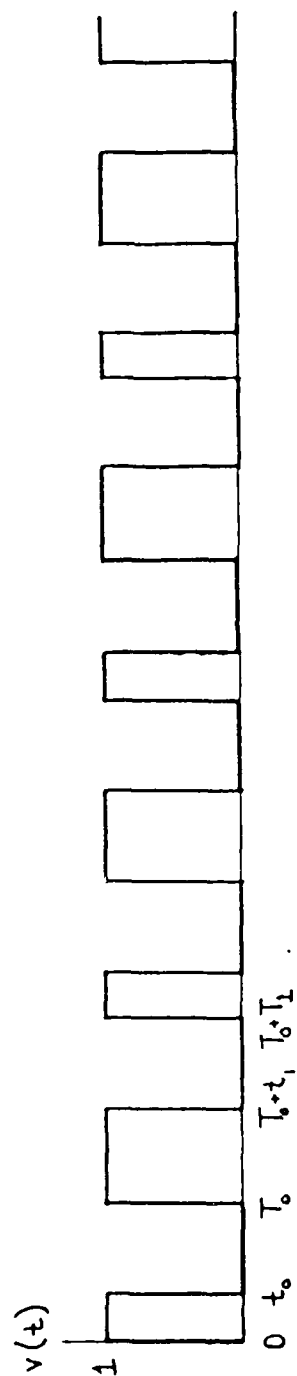
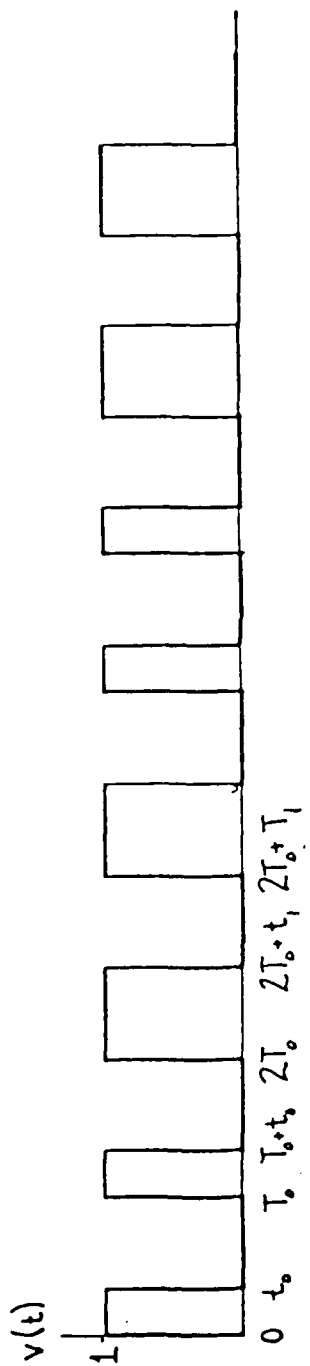


Figure 7 A sequence with two pulses.



$$\begin{aligned}
 v(t) = & [u(t - 2N(T_0 + T_1)) - u(t - (2N(T_0 + T_1) + t_0))] + \\
 & [u(t - ((2N+1)T_0 + 2N T_1)) - u(t - ((2N+1)T_0 + 2N T_1 + t_0))] + \\
 & [u(t - (2(N+1)T_0 + 2N T_1)) - u(t - (2(N+1)T_0 + 2N T_1 + t_1))] + \\
 & [u(t - (2(N+1)T_0 + (2N+1)T_1)) - u(t - (2(N+1)T_0 + (2N+1)T_1 + t_1))]
 \end{aligned}$$

Figure 8 A sequence with multiple repetitions of a pulse.

## CHAPTER 3

### TRANSFORM METHODS

#### 3.0 Introduction

How are we to describe an arbitrary pulse sequence? In the worst case we could write out each one and zero of the sequence. An approach like this takes an enormous amount of patience and is very prone to errors. A more esoteric, if not more practical, approach relies on identifying the major characteristics of the sequence and then transforming the data into a usable form. In this chapter I examine existing transform techniques for the decomposition of an arbitrary pulse sequence into fundamental building blocks. The objective is to recreate the sequence, without error, using the transformed data as input to a pulse sequence generator of some sort. The most common technique (and usually the first) used by the engineer to determine the spectral components of a signal is the Fourier transform. The procedures for this transform are well established and provide valuable insight into the composition of the input signal. The applicability of the Fourier transform is uncertain in the case of the arbitrary sequence, but the technique bears consideration. Another existing transform is the Walsh technique. This transform may prove very useful since its output is a weighted binary sequence. The Fourier and Walsh transforms will be examined in



the arbitrary pulse sequence application and are contrasted in the first part of this chapter. The rest of the chapter discusses the Walsh and Walsh related functions and transforms.

### 3.1 The Fourier series

Probably the most popular and useful transform in engineering is the Fourier transform. Chambers[6] presents an interesting method of introducing Fourier series to the reader, and it is his material which forms the basis of this section. Fundamentally, the Fourier transform takes a time-dependant function and presents the same function in the frequency domain through the relation

$$v(t) = \int_{-\infty}^{+\infty} V(f) \exp(j2\pi ft) df$$

This translation (transform) produces two different views of the same function: one picture presents time versus amplitude (i.e.,  $v(t)$ ); the other shows the frequency spectrum (i.e.,  $V(f)$ ) of the signal. The classical example is the unit height, unit width rectangular pulse centered at the origin (Figure 9). Its Fourier transform yields the  $\text{sinc}(\pi f)$  function

$$V(f) = \{\sin(\pi f)\}/(\pi f) = \text{sinc}(\pi f) \quad (\text{Figure 10})$$

As seen in Figure 10, the frequency spectrum of the transformed pulse is continuous and infinite in length. Intuitively we could reason this would happen since a perfect square wave, whose period is infinity, results from the summation of ever-increasing frequency sine waves with

amplitudes decreasing as frequency increases. In the real world perfectly square edges are difficult to produce; therefore, a practical pulse has a finite rise and fall time (i.e., rounded edges). The Fourier transform of this pulse will show that some frequencies do not exist in the  $V(f)$  spectrum or are of insufficient amplitude to cause square edges.

The discussion thus far concerns aperiodic (nonrepeating) functions. The Fourier transform for our application must account for periodic functions. Given that the function is periodic in time,  $v(t + T) = v(t)$ , then the transform for a large class of periodic functions can be expressed as a summation of weighted amplitude sine and cosine functions of increasing frequency. The frequency spectrum consists of a fundamental frequency and its harmonics. Shinnars[7] gives an excellent explanation and pictorial view of the Fourier transform of odd and even symmetry square waves. Under the constraint of being bounded with a finite number of discontinuities on the periodic interval,  $T$ , the Fourier transform series for periodic functions is given by

$$v(t) = \frac{1}{2}A_0 + \sum_{K=1 \rightarrow \infty} A_K \cos(K2\pi ft) + \sum B_K \sin(K2\pi ft)$$

$$A_K = 2/T \int_{-T/2}^{T/2} v(t) \cos(K2\pi ft) dt; K = 0, 1, 2, \dots$$

$$B_K = 2/T \int_{-T/2}^{T/2} v(t) \sin(K2\pi ft) dt; K = 1, 2, 3, \dots$$

For odd functions (Figure 11)  $A_K = 0$ , for even functions (Figure 12)  $B_K = 0$ , and the  $K$ th harmonic of  $v(t)$  is denoted by the two series terms of frequency  $K2\pi f$ ,

amplitude  $\sqrt{(A_k)^2 + (B_k)^2}$ , and phase equal to  $\tan^{-1}(-B_k/A_k)$ . In general, the Fourier transform of an arbitrary pulse sequence will contain both sine and cosine terms since the sequence generally is unsymmetrical about the time origin. Take, for example, the signal shown in Figure 7. Assume that the time origin may be placed anywhere upon the time axis and situate the signal so that one period of the sequence occurs on either side of this new origin (Figure 13). By definition, an odd function is one where  $v(-t) = -v(t)$ , and an even function is one where  $v(-t) = v(t)$ . As seen in Figure 13, there is no direct correlation between  $v(t)$  and  $v(-t)$  because of the discontinuities associated with the pulse sequence; therefore, there is no symmetry for the sequence. This does not mean there is no Fourier transform for Figure 13, but simply that the transform is more complex (i.e., both sine and cosine terms).

Let us compute the Fourier transform of Figure 13.

Let  $t_0 = 1$ ,  $t_1 = 3$ ,  $T_0 = 4$ ,  $T_1 = 4$ ;

so, the period of  $v(t)$  is  $T = 8$  and  $f = 1/T$ . For simplicity, orient the sequence as in Figure 13 and take the Fourier transform of  $v(t)$  on the interval 0 to  $T$ :

$$A_0 = 2/T \int_0^T v(t) dt = 2/8 \int_0^1 (1) dt + 2/8 \int_4^7 (1) dt = 1$$

$$A_k = 2/T \int_0^T v(t) \cos(2k\pi T^{-1}t) dt$$

$$= \frac{1}{4k\pi} \sin(4k\pi) + \frac{7/4 \sin(7(4k\pi))}{7/4(k\pi)}$$

$$B_k = 2/T \int_0^T v(t) \sin(2k\pi T^{-1}t) dt$$

$$= \frac{-4\cos(4k\pi)}{4(k\pi)} - \frac{7/4\cos(7/4(k\pi))}{7/4(k\pi)} + \frac{\cos(k\pi)}{k\pi}$$

$$v(t) = 1/2 + \sum A_k \cos(4k\pi t) - \sum B_k \sin(4k\pi t)$$

As can be seen, the Fourier transform of even a simple pulse sequence yields an infinite summation of sine and cosine functions, each with a sinc function weighting coefficient. The number of terms required depends on the tolerance imposed on the reproduced pulse sequence but may be as high as 50 to produce "crisp" edges on narrow pulses.

### 3.2 Fourier application problems

The Fourier series provides an excellent analytical tool to describe the information contained in a signal; however, the objective in this thesis is to use this description as the program input for a pulse generator. Since the outcome of the Fourier transform is a weighted infinite series of sine and cosine functions, the realization of the pulse generator requires a number of broadband frequency generators capable of generating several fundamental frequencies and  $N$  order harmonics of specific amplitudes and phases. Although this is a feasible approach, the creation of such an amplitude and phase-controllable broadband synthesizer is likely to be an expensive endeavor.

### 3.3 Introduction to Walsh functions

Digital techniques offer interesting possibilities for this application. Since the original signal is binary, a logical assumption is that a digital synthesis method will be more efficient than an analog technique. The problem still remains of how to digitally describe the pulse sequence, transform the data to a form usable by a pulse generator, and synthesize the desired sequence. The Walsh and Walsh related functions take an input signal and transform the data into a set of weighted binary sequences. As the Fourier transform produces a frequency spectrum of the transformed input, the Walsh transform produces a sequency spectrum of the input. The remainder of Chapter 3 concerns the Walsh transform and its application to the arbitrary sequence problem. Beauchamp[3] authored an excellent text on Walsh functions and their applications; his work is used extensively in the following sections.

#### 3.3.1 Sequency functions

Whereas the Fourier series use variable frequency sine waves as basis functions to describe an input signal, the Walsh and Walsh related functions use weighted sequency functions. Sequency is defined as being equal to one-half of the average number of zero crossings per unit time interval; and, sequency functions, as used by Beauchamp, refer to the non-sinusoidal signals of the Walsh and Walsh related functions. The idea is

to construct an orthogonal set of basis functions in the frequency domain with which to describe any input signal. Rectangular binary pulse sequences are used instead of the Fourier sinusoids because the synthesis of Walsh functions is done with digital equipment.

### 3.3.2 Rademacher functions

The concept of frequency functions and the creation of Walsh functions can be understood by examining the Rademacher functions. The Rademacher functions are defined on a time interval  $0 < t < T$  and are a set of increasing frequency square waves. As seen in Figure 14, the Rademacher series is ordered in terms of increasing frequency; that is, the function with the least number of zero crossings occurs first and each following function has more zero crossings than its predecessor. The arguments of the Rademacher functions are  $n$  and  $t$ ;  $t$  is time (as defined above) and  $n$  represents the order of the function. The value of  $n$  yields  $2^{n-1}$  periods of a square wave in the interval 0 to  $T$  with the peak values of the square waves equal to  $+1$  or  $-1$ . The zero crossings correspond to the zero crossings of a sine wave of angular frequency  $2^n\pi$  (i.e.,  $\sin(2^n\pi t)$ ). Rademacher functions by themselves represent a marginally useful set of functions; however, they are extremely useful in creating other functions, namely the Walsh series and Rademacher-Walsh ordering.

### 3.4 Walsh functions

Walsh functions also are defined on the interval 0 to 1 and assume the values of  $\pm 1$ , but instead of square waves (50% duty cycle) the Walsh series has variable duty cycle pulses (Figure 15). As with the Rademacher functions, the Walsh functions are shown ordered by increasing sequence values. Other orderings are possible and are discussed fully by Beauchamp[9]. Of interest in the arbitrary sequence application is the Rademacher-Walsh ordering because this set of functions can simplify digital logic design problems. Remember that the set of functions rendered by the Rademacher-Walsh ordering still is a Walsh series. The arguments of the Walsh functions are  $n$  and  $t$ ;  $t$  is time, as before  $0 < t < T$ , and  $n$  is the order of the Walsh function. The notation used for the  $n^{\text{th}}$  order Walsh function is  $WAL(n,t)$ , and actual generation of the Walsh series is accomplished by any of the four means discussed by Beauchamp[10].

### 3.5 Application of Walsh functions

Two possible uses of the Walsh and Walsh related functions exist for the solution of the arbitrary sequence problem. The first utilizes the Walsh transform, and the second uses the simplifying properties of the Rademacher-Walsh ordering.

### 3.5.1 Walsh transform

Similarities exist between the Fourier and Walsh transforms. Both produce a summation of orthogonal basis functions; the Fourier series consists of sine and cosine functions while the Walsh series contains N order of Walsh functions. The Walsh transform is written in the form

$$v(t) = a_0 \text{WAL}(0,t) + \sum a_n \text{WAL}(n,t)$$

$$a_0 = 1/T \int_0^T v(t) \text{WAL}(0,t) dt$$

$$a_n = 1/T \int_0^T v(t) \text{WAL}(n,t) dt$$

$$n = 1 \rightarrow \infty$$

The number of terms required for reproduction of the input signal depends on the acceptable magnitude of the mean-squared error (MSE) and the characteristics of the input signal with respect to those of the transformation series. Given that the input signal is a rectangular pulse sequence, I would expect the Walsh transform to yield a shorter transform series than the Fourier transform. This is because the Walsh transform does not have to use an infinite series to represent a rectangular pulse (see Section 3.1). As an example, let us use the waveform of Figure 13 as the input to the Walsh transform and refer to Figure 15. As with the Fourier transform, the weighting coefficients must be computed by an integration over the interval T. Since the input signal is equal to zero for portions of T, the integrands are simplified.

$$a_0 = 1/8 \int_0^8 v(t) \text{WAL}(0,t) dt = 1/8 \int_0^1 (1)(1) dt + 1/8 \int_4^7 (1)(1) dt$$
$$a_0 = 1/2$$



In a similar manner,  $a_1$  through  $a_7$  can be computed and the values found as follows:

$$\begin{aligned} a_1 &= 1/4 & a_2 &= 0 & a_3 &= 1/4 & a_4 &= 0 \\ a_5 &= -1/4 & a_6 &= 0 & a_7 &= 1/4 \end{aligned}$$

With these weights we can now describe the input in terms of the Walsh transform.

$$v(t) = (1/2)WAL(0,t) + (1/4)WAL(1,t) + (1/4)WAL(3,t) - (1/4)WAL(5,t) + (1/4)WAL(7,t)$$

When these functions are summed, we find that the Walsh transform has produced an exact replica of the input signal ( $MSE = 0$ ). The order of Walsh functions required for  $MSE = 0$  is discussed in the following section. Therefore, the Walsh transform produced a finite number of terms for Figure 11, where the Fourier transform produced an infinite series. This example illustrates the significant difference in outcome between the Walsh and Fourier transforms when applied to a binary pulse sequence.

### 3.5.2 Walsh transform problems

Although the Walsh transform seems ideally suited for the arbitrary sequence problem there are several hidden problems. To compute the weighting constants, the user must piecewise integrate over the interval  $T$ . Although this is not difficult, the process can become extremely tedious and prone to error. This makes these calculations prime candidates for computer processing; except, we must now somehow specify to the computer

what the input signal looks like. As seen before, the expression of  $v(t)$  using unit step functions becomes very complex even for simple pulse sequences, and if we specify the pulse train by ones and zeroes, why not simply program the computer to sequence through memory and generate the pulse train in this manner without resorting to Walsh transforms? This problem has the form of the 'chicken or the egg' debate!

Presuming that a suitable means exists to obtain the Walsh transform, there is another hazard. The Walsh function creates an exact replica of the input signal only if for every zero crossing of the input there exists within the set of Walsh functions of order  $2^n = N$  a zero crossing at the same instant of time. This is a problem where the interval,  $T$ , is not a multiple of  $N$ . Take, for example, the case where the PR equals  $(1/3)T$  (Figure 16). The zero crossing for the transition from ON to OFF occurs at  $t = 1/3$ , and since  $1/3$  is not evenly divisible by  $2^n$ , only at  $n = \infty$  will the Walsh functions converge to create a zero crossing at  $t=1/3$ . This is because the Walsh series is derived from the Hadamard functions; these functions create zero crossings which double in frequency for each increase in order (refer to Section 3.4). As seen in Figure 16, the error manifests itself at the zero crossing. There is an amplitude error and a zero crossing timing error which both reach zero MSE for  $n = \infty$ . Since the sequence designer knows that the amplitude values must ultimately be either zero or one, an error threshold can be

implemented for the amplitude error; but, the zero crossing error may not be eliminable except for high orders of Walsh functions.

Even if the cost of the Walsh generators were so low that many could be placed on a single chip, there is still the fundamental limitation that the order of the Walsh functions essentially quantizes  $T$  into  $2^n$  pieces, and if the input function is quantized more finely, more Walsh function generators must be built. Some mention should now be made about the MSE as it applies to the Walsh transform. A reasonable termination point for the Walsh transform is when the series summation for each PW of the synthesized sequence is accurate to within a rise/fall time margin. The computation for this MSE must be made individually for each edge of the pulse sequence: again, a tremendously tedious task.

### 3.6 Rademacher-Walsh transform

Several excellent papers have been published on this topic. Edwards' [11] article is most appropriate for this discussion, but the reader should refer also to Bennett[12], Picton[13], Haring[14], and Chow [18] for more on this topic. Essentially, the concept is that any Boolean function can be constructed from a set of exclusive-OR gates and optimized universal threshold gates. If you looked at the desired pulse sequence as a series of ones and zeroes, the method of the Rademacher-Walsh transform yields spectral coefficients which

are then ordered as Chow parameters and then cross-referenced to tables containing appropriate weighting information for the threshold gates. The problems associated with this method are identical to the Walsh transform with one addition. The current state-of-the-art in threshold logic precludes widespread production of the arbitrary level threshold gate. The tolerances required for accurate determination of threshold crossing are quite strict when more than two levels are permitted (i.e., ordinary AND and OR gates are special cases of threshold gates). If any integer weighting is allowed, the threshold gates become extremely difficult to produce.

### 3.7 Chapter summary

Although the Fourier transform is well developed, in the arbitrary pulse sequence problem this transform is clumsy in its rendition of the desired pulse sequence. The Walsh transform produces exact replicas of the input signal if we are given enough generators to adequately quantize the time interval  $T$ . Even if there are sufficient numbers of generators, we still face a big problem in describing the input sequence to the Walsh transform generator and in synthesizing sequences quantized finer than the highest order Walsh function generator. The techniques of the Rademacher-Walsh transform offer the possibility of exact reproduction were it not for the electrical difficulties of threshold gate production. Even as technology produces reliable threshold gates, the

Rademacher-Walsh transform suffers the same pitfalls as the Walsh transform. So, there does not seem to be a distinct solution from the methods examined in this chapter; therefore, other methods must be investigated.

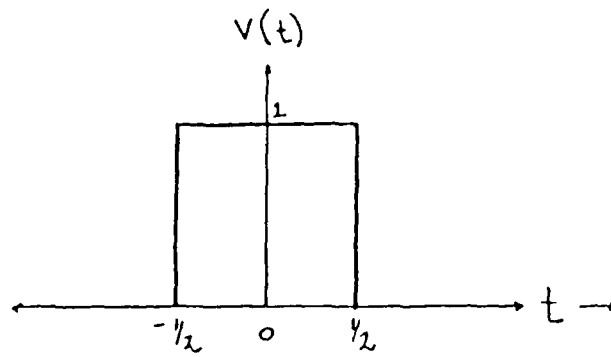


Figure 9 A unit height, unit width pulse.

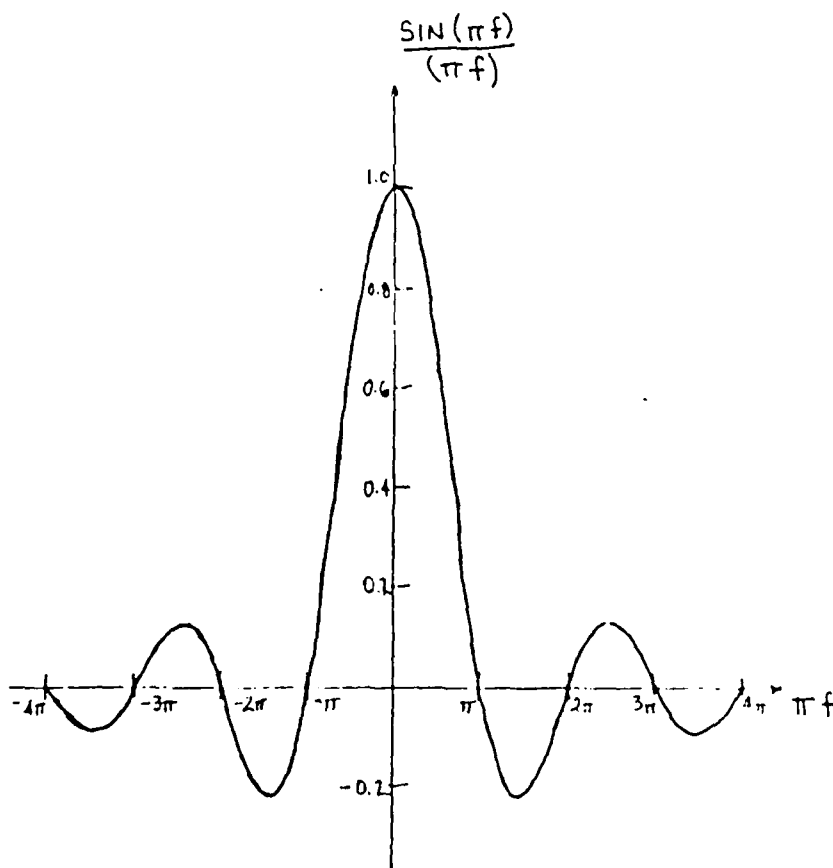


Figure 10 The sinc ( $\pi f$ ) function.

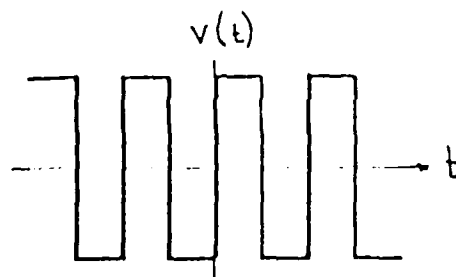


Figure 11 A function with odd symmetry.

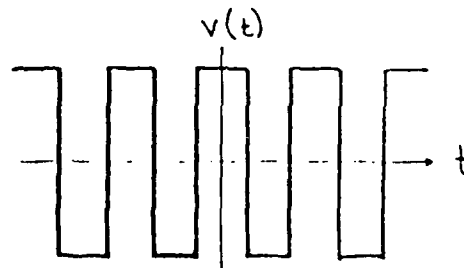


Figure 12 A function with even symmetry.

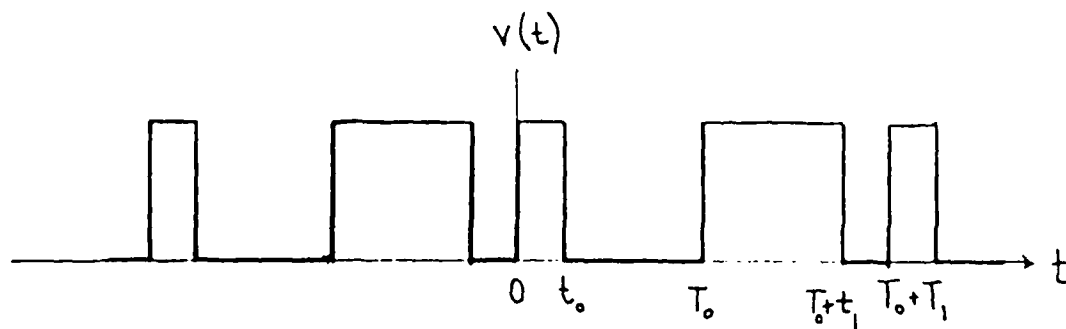


Figure 13 An asymmetrical pulse sequence.

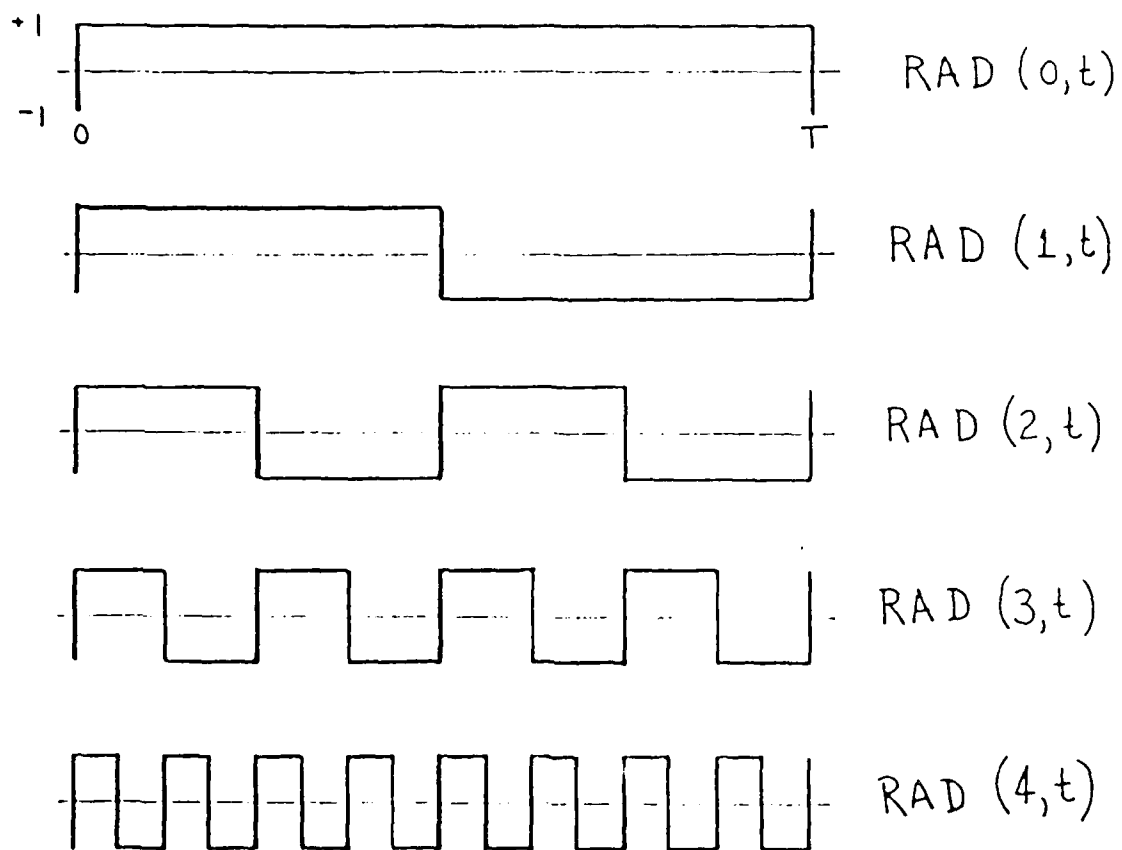


Figure 14 A set of Rademacher functions for  $N = 4$ .



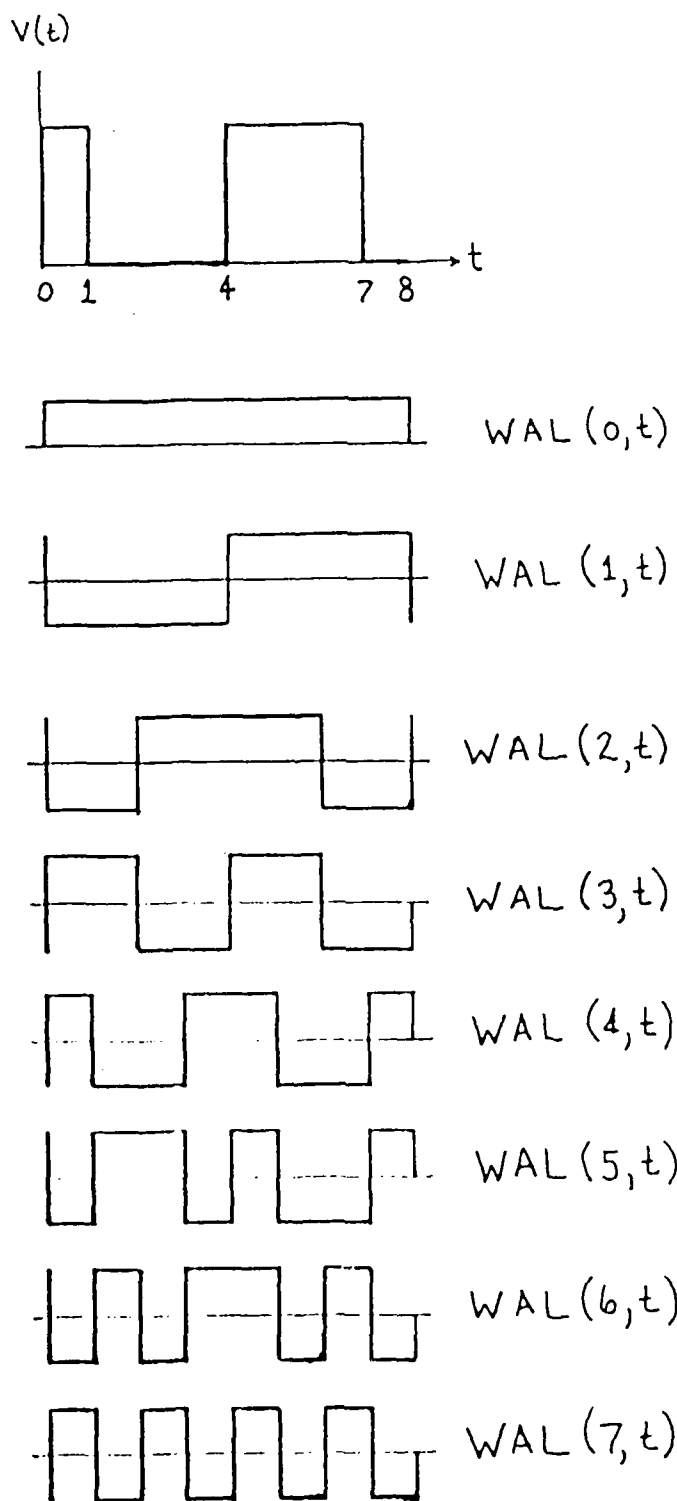
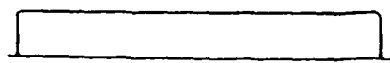
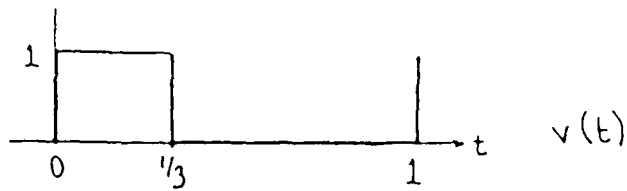


Figure 15 A set of Walsh functions for  $N = 8$ .



$WAL(0,t)$

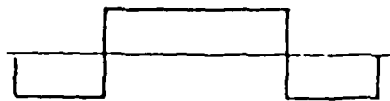
$$a_0 = 1/3$$

$$a_n = \frac{1}{T} \int_0^T v(t) WAL(n,t) dt$$



$WAL(1,t)$

$$a_1 = -1/3$$



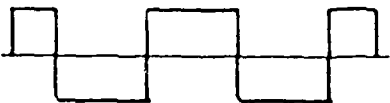
$WAL(2,t)$

$$a_2 = -1/6$$



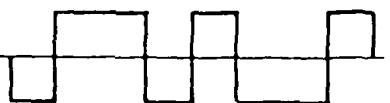
$WAL(3,t)$

$$a_3 = 1/6$$



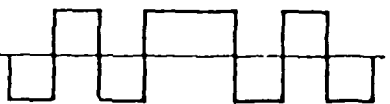
$WAL(4,t)$

$$a_4 = -1/12$$



$WAL(5,t)$

$$a_5 = 1/12$$



$WAL(6,t)$

$$a_6 = -1/12$$



$WAL(7,t)$

$$a_7 = 1/12$$



Figure 16 The Walsh transform of  $v(t)$ .

## CHAPTER 4

### HARDWARE AND SOFTWARE METHODS

#### 4.0 Introduction

In this chapter I will examine two methods to synthesize the desired arbitrary pulse sequence. Computers have permeated virtually every aspect of engineering; so, why should this application be any different! Logically, you would expect a computer to make quick work of this arbitrary sequence problem, but I will demonstrate otherwise. A hardwired solution seems attractive, but there are significant problems associated with making an all-purpose hardware-oriented pulse sequence generator.

Before proceeding, a word on how the pulse sequences are examined in this chapter is appropriate. In Chapter 3 the Walsh transform produced a perfect reproduction of the input signal only when the Walsh functions had quantized the interval  $T$  finely enough to produce a zero crossing at the same instant  $v(t)$  had a zero crossing. This 'matching the edges' concept is used here to describe the pulse sequences. For each sequence, find the smallest time interval which, when weighted with a zero or one, can be 'summed' (actually, concatenated) to exactly reproduce the pulse sequence. In Figure 13, the smallest time interval is  $(1/8)T$ , and when weighted with ones and zeroes, the sum of similarly weighted quantizations equals

the original pulse sequence.

$$v(t) = (1)(1/8) + (0)(1/8) + (0)(1/8) + (0)(1/8) + \\ (1)(1/8) + (1)(1/8) + (1)(1/8) + (0)(1/8)$$

Algebraically, this equation is not correct, i.e., the result of the equation is  $v(t) = 1/2$ ; a nonsensical statement!), but only concepts are being stressed here.

#### 4.1 A computer solution

The computer offers the possibility of programmability in constructing the pulse sequence. Ideally, a high-level language specifically for the construction of sequences would give people other than the machine language programmer the ability to create arbitrary pulse sequences. There are basically two problems with using computers for this application: 1) how do you specify the desired sequence to the computer (see Section 3.6.1), and 2) what do you do about the speed degradation incurred from the software overhead? If the desired sequence can be characterized by an algebraic equation, then the computer simply computes the placement of the zero crossings on the time axis and executes a timer routine to count these intervals. As seen in Chapters 2 and 3, very few of the pulse sequences are described in a closed and convenient algebraic form suitable for computer generation. When equations fail to describe the problem, look-up tables are usually the answer. So, now the computer is relegated to the mundane task of looking in a table for the value of  $v(t)$  as time is counted. Let us examine this look-up table: without

some form of compaction, every one and zero in the sequence must be listed for each quantized interval of  $T$ . For short pulse sequences, as shown in Figure 15, this means only eight bits of memory are required; however, for an application such as radar where the PRIs are in the hundreds of microseconds and the PWs are on the order of a few microseconds, a tremendous amount of memory is required for the look-up table.

Even if there is no problem with wasting memory, the question remains of whether the computer has a large enough operating bandwidth to handle narrow pulses. Of the faster micro-computers priced around a few thousand dollars, most have master clocks whose frequencies are less than 15 MHz. This restricts their CPU cycle times to the order of a hundred nanoseconds at best. Even with the software resident in firmware microcode, several instructions must be executed for each pulse, thus creating a software cycle time in the few microseconds range. Under this time constraint, synthesis of a 0.1  $\mu$ S time interval is impossible. Using a pipeline improves the system throughput, but the pipe's output can not be clocked any faster than the CPU can fill the first section of the pipe. The solution to the speed problem is to spend tens of thousands of dollars for a fast system or reduce the available bandwidth imposed on the desired pulse sequence.

#### 4.2 A hardware solution

Provided that no algebraic method is suitable for

description of the arbitrary sequence, the next logical solution is to hardwire a circuit to synthesize the signal. This method works very well until a revision is needed in the sequence. Therefore, the problem is to create a generic pulse synthesizer. Shift register technology seems appropriate in this situation. Golomb[15] authored a text on shift register sequences, and his work is the basis for the following discussion.

We first begin with an  $n$ -stage feedback shift register (Figure 17) with  $x_n$  being the highest order stage and  $F(x_1, x_2, \dots, x_n)$  representing the feedback function. Also, allow for an initial value for each stage, (i.e., the parallel load inputs) and assume that each stage can hold only the value zero or one. The operation of the shift register is simple; on command each stage places its current value into the next higher order stage; that is,  $x_{k+1} = x_k$ . The first stage's input comes from the output of the feedback function, and the value contained in  $x_n$  is shifted to the outside world. Usually, the command to shift originates from a system clock, but this need not be the case.

With  $n$  stages and only ones and zeroes allowed in each stage, there exist  $2^n$  different possible combinations of binary values distributed over the  $n$  stages. For example, if  $n=2$ , then there are  $2^2 = 4$  possible combinations for the two stages:

$x_1$	$x_2$	$S_k$
0	0	$S_0$
0	1	$S_1$
1	0	$S_2$
1	1	$S_3$

As shown above, there are four distinct combinations for the two stages; label each individual combination the 'state' of the shift register and call it  $S_k$  for  $k = 0 + 2^n - 1$ . Now, each state uniquely identifies the corresponding values held in each stage. The order in which the shift register sequences through the possible states depends on the initial conditions and the feedback function,  $F$ . If  $F$  calculates the next state from the initial state through the relation

$$S_n = c_1 S_{n-1} + c_2 S_{n-2} + \dots + c_r S_{n-r}$$

$c_1, \dots, c_r = 0$  or  $1$  (the value is independent of  $n$ )

then the shift register is a linear shift register and the relationship is that of linear recurrence. In the arbitrary sequence problem, the pulse sequence typically follows no recurrence relationship; that is, the value of the next pulse is independent of the current value. If a recurrence relationship exists, then determining the feedback function required to produce a given sequence is relatively easy and the methodology straightforward. Under the constraints of linear recurrence, there exists  $2^r$  different state sequences for  $y = 2^n$ , and the maximum length of a single sequence is bounded to  $2^n - 1$  states. For linear recurrence, the state  $S_0$  (all zeroes) does not occur. So, even if

the desired pulse sequence could be computed by linear recurrence, the absence of the all zeroes state is a fatal shortcoming. Again, refer to Figure 13; the sequence for this pulse train is 1,0,0,0,1,1,1,0. Here, there are eight time slices where the pulse train can assume either zero or one. Three binary bits uniquely specify eight states; therefore, a 3-stage shift register could be used. Assume for a moment that this sequence is a linearly recurring string; then, the synthesis of the 0,0,0 portion of the signal is impossible due to the absence of  $S_0$ .

If the restriction of linear recurrence is lifted, then the shift register can synthesize a total of  $2^n$  unique state sequences where  $z = 2^{n-1} - n$ . This large increase of possible state sequences is due to lifting the restriction that the next state be predictable from the current state. This is called nonlinear shift register logic and is quite applicable to the arbitrary sequence problem. There is, of course, a penalty paid for the nonlinear logic; Golomb[16] has proven conclusively that for the nonlinear case there is no simple algebraic means of creating the appropriate feedback logic, given the desired state sequence. This is in direct contrast to the linear feedback case where several methods are obtained for easy generation of the appropriate logic when given a desired sequence. For a small sequence, the logic generation problem is no more difficult than finding a minimal sum-of-products form of a 2, 3, or 4 variable function. As an



example refer to Figure 13 for the pulse sequence and Figure 17 for the case of  $n = 3$ ; the sequence describing this pulse train is 1,0,0,0,1,1,1,0 . and as stated above, a 3-stage shift register has eight possible states. Shifting to the right, the state sequence would look like this:

$x_1$	$x_2$	$x_3$	output	next input to $x_1$
0	0	1	initial state	0
0	0	0	1	1
1	0	0	0	1
1	1	0	0	1
1	1	1	0	0
0	1	1	1	1
1	0	1	1	0
0	1	0	1	0
0	0	1	0	

The initial state is given by the first three time slices (quantizations) of the desired signal, and each shift command forces a bit out of  $x_3$  representing the value of the pulse sequence for that time slice. So, after the first shift, the output equals one; after the second shift the output equals zero, and so on. Since Golomb[17] has shown that shift register state sequences are periodic, the output of this shift register is the desired pulse sequence. But how does one develop the necessary feedback logic? The state sequence for this pulse sequence is developed with the next value of  $x_1$ . If we use  $x_1$ ,  $x_2$ , and  $x_3$  as input variables to the feedback function and  $F$  (the next state of  $x_1$ ) as the output of the combinational logic network we get the following truth table:

$x_1$	$x_2$	$x_3$	$F$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$F = x_1 \bar{x}_3 + \bar{x}_2 \bar{x}_3 + \bar{x}_1 x_2 x_3$$

$$F(x_1, x_2, x_3) = \Sigma(0, 3, 4, 6)$$

The function  $F$  was simplified using a 3-variable Karnaugh map and is represented in literal form as well as decimal form. Realization of  $F$  occurs through AND, OR, and NOT gates connected according to the literal form of  $F$ . Also, a decoder could detect the presence of the true states in decimal form and produce a true output. The decoder form of  $F$  is preferable since only output connections need to be changed for a change in pulse sequence.

#### 4.3 Problems with the hardware solution

There is a fundamental problem with the shift register solution: if the arbitrary sequence is quantized more finely than the shift register system, the shift register synthesizer will have a noneliminable error (MSE is nonzero). Further, if there are a large number of quantizations, the shift register may be 10, 20, 30, or more stages in length. Although it is relatively easy to create a single integrated circuit with a

large number of shift stages, the problem occurs of how to create a decoder with that number of inputs. Realistically speaking, there are many sequences with periods in the several minutes' time frame which require  $0.1 \mu\text{s}$  quantizations. If the period were just 2 minutes long, there would be  $1.2 \times 10^7$  quantizations requiring 31 stages. Now the decoder must have 31 inputs and  $1.2 \times 10^7$  outputs. This is unrealistic for a decoder, so the feedback function would have to be realized by vertex gates. Now, five and six variable equations are difficult enough to reduce, let alone a 31 variable function. Generally speaking, the user would have to accept some error in the signal synthesis.

#### 4.4 Chapter summary

Walsh and Fourier transforms had shortcomings in the arbitrary sequence problem, so other methods of synthesis were examined. A computer would allow great flexibility in the creation of an arbitrary sequence if the sequence could be described algebraically in a programmable form. An arbitrary sequence has no special algebraic description. Moreover, if a sequence could be programmed, the time spent in software overhead would restrict the smallest time interval synthesizable to some value depending on instruction execution times. The user would then have to decide if this were acceptable.

If a computer can not be used, perhaps a hardware device

can generate the sequence. Shift registers have been used in the past and seem a likely candidate for this application. Since the requirement is to generate any sequence, the linear shift register model is inappropriate for this use. This means that nonlinear shift register theory must be used; and, Golomb has proven that no single, efficient algebraic means exist which can easily describe the nonlinear feedback function required for a specified pulse sequence. For many applications, the nonlinear shift register proves to be an effective arbitrary sequence generator; however, if the sequence to be synthesized is quantized more finely than the shift register system, there will be errors in the output waveform. Also, if there are a large number of quantizations, the feedback network becomes impractical.

So, we have a candidate for the synthesizer but still no effective means of describing and creating a truly arbitrary pulse sequence.

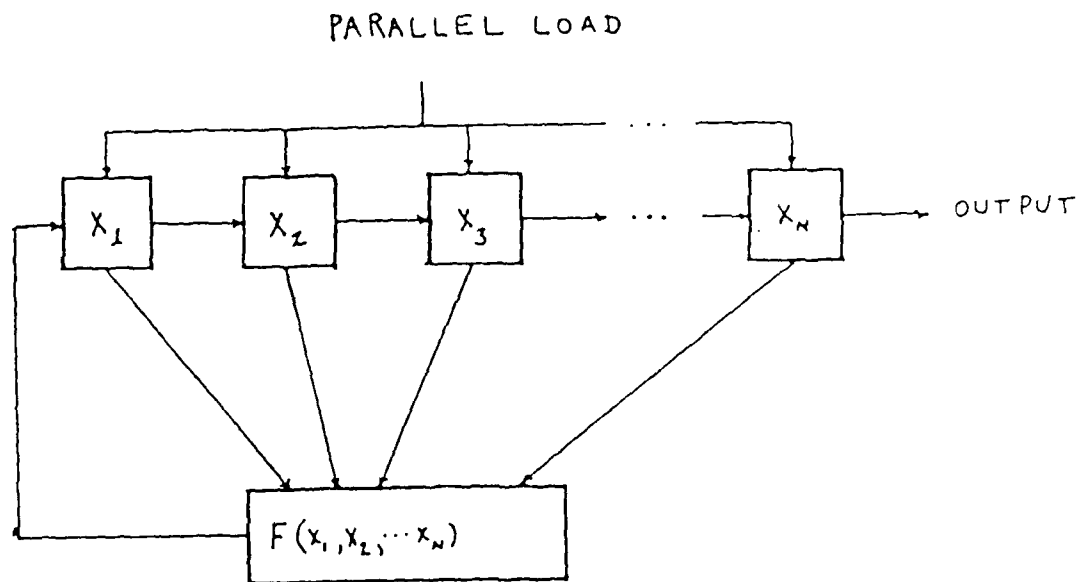


Figure 17 An n-stage, parallel loaded, feedback shift register.

## CHAPTER 5

### DESCRIPTION OF AN ARBITRARY SEQUENCE

#### 5.0 Introduction

Of the material covered thus far, it seems to me that there exists some commonality between the different techniques' shortcomings. For each of the methods examined there always was the problem of how to easily describe the desired pulse sequence. In each case the user was forced to break the sequence into individual ones and zeroes and then proceed with the transform or the feedback function synthesis. Except for special cases, no algebraic description could predict the parameters of the next pulse in the sequence because of the nature of the arbitrary sequence. Usually the sequence performs some function and therefore probably has a mathematical description, but the objective is to describe and generate an arbitrary sequence.

The second point of commonality concerns the quantization level. Each method required the knowledge of how many pieces the periodic interval  $T$  must be broken to completely specify the pulse sequence. The problem is that given a sequence of interval  $T$  quantized to some degree, there always exists another sequence which requires a finer quantization. To achieve a finer quantization for the Walsh transform the user must provide higher order Walsh function generators. In the case of the shift registers, more stages are required for a

finer quantization.

This chapter presents my proposals for a pulse sequence description and solution of the quantization level dilemma.

### 5.1 Quantization solution

The problems of pulse sequence description and quantization level should not be considered as independent of one another. Earlier I alluded to the 'real' world in that a sequence is not truly arbitrary since the sequence designer probably had some function in mind for the pulse train. I now state that the designer must also account for the method in which the sequence will be synthesized. As seen in the Fourier and Walsh transform comparisons, a digital synthesis is likely to be more compact (efficient) than an analog synthesis. With this in mind, the sequence designer realizes that the pulse generator has a finite bandwidth; that is, there is a master clock somewhere in the system which limits the minimum quantizable time interval. Let the smallest interval where we can only have one quantization (i.e., this is an interval which can not be subdivided into smaller time slices by the system) be denoted as  $q_k$ . This interval,  $q_k$ , typically equals one master clock cycle but could be equal to several clock 'ticks,' depending on the device. Mathematically, let  $q_k$  be any real number bounded by zero and infinity:

$$0 < q_k < \infty$$

Now, impose the restriction that the sequence's interval

of periodicity,  $T$ , be an integer multiple of  $q_k$ :

$$T = N \times q_k$$

Another way of stating this restriction is to say that every PW and PRI in the sequence must be an integer multiple of  $q_k$ . Notice that nothing is said about whether or not the actual system could realize a particular value of  $q_k$ . Mathematically, you can foresee a case where each pulse's quantization was referenced to a different  $q_k$  value. In general, each PW and PRI - PW time interval in the pulse sequence could be referenced to a different  $q_k$  value. The extreme limit for this idea is that within a PW or PRI - PW time interval are different  $q_k$  values; and although this is feasible, the definition of  $q_k$  excludes this possibility. Since  $q_k$  is a real quantity, every interval can be constructed from an integer multiple of a real number; therefore, only one value of  $q_k$  is needed per PW or PRI - PW interval. This concept allows for the case where the pulse generator switches between reference clocks or the instance where several distinct generators produce a composite pulse sequence.

Now, the interval  $T$  takes on a new look. The entire sequence appears as a summation of individual  $q_k$ 's, and the total time required to produce every pulse gives the value of the sequence's periodic interval,  $T$ , and the number of quantizations. Previously, we looked at the interval  $T$  and attempted to divide it into equal slices of time, but now the



idea is to build each pulse individually from its own quantization value. From this idea comes a pulse synthesizer which creates each pulse individually without regard to the value of  $T$ . This value of  $T$  will come as a 'matter of fact' from the summation of the sequence's pulses.

## 5.2 The pulse description

Based on the results of Section 5.1, we must now describe the sequence with respect to its quantization values for the  $k^{\text{th}}$  pulse's PW and PRI - PW time intervals. Let  $x_k$  be the quantization value of the  $k^{\text{th}}$  PW, and let  $y_k$  be the quantization value of the  $k^{\text{th}}$  PRI - PW. Referring to Figure 18, let the  $k^{\text{th}}$  pulse be denoted as  $P_k$  and described as follows:

$$P_k = 4x_k \text{ )+( } 3y_k$$

$$\text{where } x = q_1, \quad y = q_2 \text{ and } q_1 \neq q_2$$

The operator  $\text{ )+(}$  serves two functions: 1) it provides for the algebraic sum of the PW and PRI - PW time intervals; and, 2) it provides a notation showing that there exists an ordered relationship between  $x_k$  and  $y_k$ . The total time required for a pulse (i.e., the PRI) is found by summing the ON and OFF times; so, we have the relation

$$\text{PRI} = \text{ON} + \text{OFF} = \text{OFF} + \text{ON}$$

$$\text{PRI} = 4x_k + 3y_k = 3y_k + 4x_k$$

This is the standard use of the '+' symbol, and all properties of addition hold under the symbology of this

application. The problem arises when trying to describe the shape of the pulse. Normally, the ON time precedes the OFF time:

$$P_k = \text{ON})(\text{OFF}$$

The notation  $)()$  denotes concatenation of two quantities: the right-side quantity is concatenated to the end of the left-side quantity. In this application,  $)()$  symbolizes that the OFF time immediately follows the ON time. However, there is no reason why an OFF time should not precede an ON time:

$$P_k = \text{OFF})(\text{ON}$$

Since in any pulse sequence there exists an algebraic summation to determine  $T$  and a concatenation of ON and OFF times, I create the operator  $)+($  with the following properties:

1. The period  $T$  is computed from the standard algebraic sum of the quantities specified (ignore the  $)()$  symbols). In a similar fashion, the PWs and PRIs are computed from the individual terms.
2. The ON/OFF sequence of the signal is ordered according to the concatenation shown by the operator (ignore the  $+$  symbol). Refer to Figures 18 and 19 to see that

$$P_k \neq P_{k+1}$$

that is,  $4x_k)(3y_k \neq 3y_k)(4x_k$

$$T = 4x_k + 3y_k = 3y_k + 4x_k$$

$P_k$  is ordered as in Figure 18

$P_{k+1}$  is ordered as in Figure 19

$$P_1 = P_2 \text{ iff}$$

$$T_1 = T_2$$

AND every concatenation matches exactly.

Essentially, I have created an algebraic concatenation operator; with a single symbol the composition of the pulse sequence is completely specified. An extremely important reminder is now apropos. Remember that every sequence studied

thus far is periodic; therefore, the notations given above are periodic as well. This means that Figures 18 and 19 represent infinite series described as shown above.

### 5.3 Capitalize on unnecessary repetition

The concept of time interval redundancy is implicit in the derivation of the new notation in Section 5.2. If a PW interval is quantized to some degree, the shorthand description specifies this interval as an integer multiple of some base quantization. Let us use this concept for the entire pulse sequence. Whenever there are identical adjacent pulses, specify this portion of the sequence as an integer multiple of that single pulse. Referring to Figure 20, I derive the following description:

$$\begin{aligned}
 P &= P_1 )+( P_2 )+( P_3 )+( P_4 \\
 P &= 3[x_1 ]+( 2y_1 ] )+( [2x_1 ]+( 2y_1 ] \\
 &\quad \text{for } x_k = y_k = q_0 \text{ and} \\
 &\quad P_1 = P_2 = P_3 \\
 P &= 3P_1 )+( P_4
 \end{aligned}$$

In this example, the brackets signify a 'unique' pulse, and the integer bracket multiplier specifies the number of repetitions of that unique pulse. A unique pulse becomes a basis 'vector' in the construction of the sequence. Within a given sequence there may be some pulse,  $P_k$ , whose parametric values and concatenation order appear again somewhere within the sequence (i.e.,  $P_{k+j}$ ). The task is then to identify all unique pulses in the desired sequence,

declare the number of pulse repetitions of each should there be identical adjacent pulses, and order the equation for  $P$  so that the concatenation order is correct. For the above example (Figure 20), there are two unique pulses in the series. Since there are three duplications of  $P_1$ , the first part of the equation for  $P$  becomes

$$P = 3(x_1) + (2y_1) + (\dots)$$

$$P = 3P_1 + (\dots)$$

With only one unique pulse remaining in the sequence,  $P$  becomes specified, as shown previously.

#### 5.4 Block repetitions

In the general case, there may exist portions of the arbitrary sequence which repeat: that is, a group or block of unique pulses repeat some number of times before proceeding on to the next pulse of the series. Referring to Figure 21, suppose there is a single block consisting of two unique pulses: that block repeating itself twice before proceeding on through the sequence. The sequence description appears as follows:

$$P = 2(x_1) + (2y_1) + (4x_2) + (y_2) + (2x_3) + (y_3) + (2x_4) + (y_4)$$

$$P = 2(P_1) + (P_2) + (P_3) + (2P_4)$$

In this example, there are four unique pulses: one block repeating twice with two unique pulses, a single repetition of the third unique pulse, and two repetitions of the fourth unique pulse. If you were forced to specify this sequence

without the shorthand notation, the sequence of 20 binary digits appears like this:

1,0,0,1,1,1,1,0,1,0,0,1,1,1,1,0,1,1,0,1,0

From a mathematical standpoint, there is no limitation on the level of nesting for the block notation; therefore, a given sequence could have a block nested within a block which is nested within a block of pulses, and so on.

### 5.5 The random sequence

The assumption that the pulse sequence was periodic produced a succinct description for an arbitrary pulse sequence, but what of the case where the function is aperiodic? Consider a case where the sequence is defined exactly for some interval of time but enters a random mode for a finite period of time. This function still has attributes of a periodic signal if you look at the 'big' picture. Using the previous notation for periodic pulse sequences we derive an equation of the form:

$$F = \{ \text{[ specified portion of signal ]} \} + ( \text{RANDOM(PW,PRI,PLS REP,BLK REP, MISC INPUTS)} )$$

The known (explicitly stated) part of the signal uses the regular method of description, and the random portion is signified by the function RANDOM. RANDOM is a function whose outputs are the PW, PRI, pulse repetitions, and block repetitions of a random (in fact, pseudorandom because of finite time) sequence of pulses determined by the random function and its set of inputs. Of course the assumption that

the sequence is periodic is invalid for the random case, but you can see how development of the periodic sequence has allowed for some quantification of the random sequence description.

#### 5.6 Chapter summary

An examination of the 'real' world constraints imposed on the generation of an arbitrary sequence yielded a succinct form to describe these sequences. The key lies in the fact that most pulse sequences consist of redundant time information. By assuming a periodic sequence, the shorthand notation exactly and compactly describes the sequence. By allowing for repeating blocks of unique pulses and random time intervals, the sequence description is capable of describing virtually any arbitrary sequence.



Figure 18 A pulse for  $P_1 = 4x_1 + (3y_1)$ .

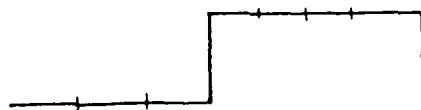


Figure 19 A pulse for  $3y_1 + (4x_1)$ .



Figure 20 The sequence for  $3P_1 + (P_2)$ .

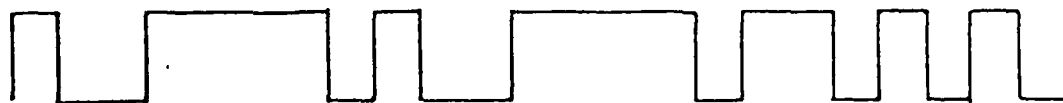


Figure 21 A pulse sequence containing blocks.

## CHAPTER 6

### IMPLEMENTATION OF THE PULSE DESCRIPTION

#### 6.0 Introduction

A pulse sequence description is only as good as its implementation. If the description provides for a quicker transform, then the description is useful; if an arbitrary pulse synthesizer is based on this description, then the notation is again useful. In Chapter 5, I developed a shorthand notation for describing an arbitrary pulse sequence. Now let us focus on the problem of generating such a sequence from this notation. The first part of Chapter 6 deals with a generic pulse generator in a block diagram fashion. The remainder of this chapter concerns my speculation of what is required to implement these ideas in firmware and software.

#### 6.1 Characteristics of the arbitrary sequence

From the previous chapters, the assumption is made that the desired pulse sequence is periodic; and, for the situation where randomness occurs, there is an extension allowed in the shorthand notation. Use of time redundancy yields a description which is compact and exactly describes the sequence. So, to summarize Chapter 5, any sequence has the following characteristics:

1. There exists an ordered sequence of ON and OFF times, the order being specified by the operator  $\rightarrow$  or  $\leftarrow$ .
2. Each ON and OFF time is composed of some integer number  $n$  of



quantizations, the quantizations possibly being different for the various ON and OFF intervals.

3. A unique pulse is one where its PW and PRI - PW values are different from all other pulses of the same order in that sequence (i.e., the same ON/OFF or OFF/ON ordering).

4. When adjacent pulses are identical, that portion of the sequence is specified by an integer multiple of that unique pulse.

5. When there is a group (block) of pulses which repeats several times before the pulse sequence resumes, the pulses comprising the block are grouped together, and the number of block repetitions becomes the integer multiplier of the specified block. Theoretically, there is no limitation to the nesting level allowed for block repetitions.

6. For the case where a sequence contains random pulses, the function RANDOM is inserted in the sequence description at the point where the randomness begins.

7. The entire sequence is presumed to be periodic with a value equal to the summation of all PW and PRI - PW values.

## 6.2 A generic pulse generator

Based on the results shown in Section 6.1, an implementation of the sequence description is found in this section. Only block diagram (Figure 22) format is considered for this discussion; this allows the generator described to have an infinite operating bandwidth in terms of memory and speed. If an arbitrary sequence is described by the equation

$$P = a_1 P_1 + (a_2 P_2) + ( \dots ) + ( n(a_k P_k) + (a_{k+1} P_{k+1}) ) \\ \text{for } P_k = n_k x_k + (m_k y_k) \\ n_k, m_k, a_k = \text{integer}$$

then the pulse generator must be capable of counting quantization intervals, pulse repetitions, and block repetitions, and the system must recognize blocks of pulses and the beginning and end of the specified sequence. Binary

counters are used to count the quantization intervals, pulse repetitions, and block repetitions, and control logic is required to recognize state conditions within the system. Some form of memory is required to store the data, and since the data are different with regard to the PW/PRF - PW time intervals, pulse repetitions, and block repetitions, at least three separate memory divisions are required. Last, some form of interfacing to the outside world for the output is required.

But how would such a device use this information? Since the desired pulse sequence is finite in length (for the periodic case, at least), the user must recognize the beginning and end of the sequence. From the literature on computer software, I call the complete sequence a "file" of pulses because this information will become the program for the generic pulse generator. Once the file has been identified (usually a very easy task), the sequence designer must identify any blocks of pulses within the file. Next, find any portion of the signal which has multiple repetitions of a unique pulse, and finally, the only items left will be single repetition pulses. As with the equation describing the sequence, order the pulse information from the first pulse to the last pulse of the file. For example, examine the pulse sequence shown in Figure 20: there are two unique pulses in the series producing a total of eight ON/OFF time intervals, and only the first unique pulse has multiple repetitions. The ON time of  $P_1$  represents the first time interval of the series, and the OFF

time of  $P_4$  is the last time interval of the file. Note that in this example there are no blocks of pulses.

To operate the generic pulse generator we would place this data into the memory of the system and execute the program. The program might look like this:

MEM #	INT VAL	PLS REPS	BLK REPS	CMD
0	1	3	0	FSA
1	2	3	0	NOF
2	2	1	0	NOF
3	2	1	0	EOF

FSA => File Start Address      EOF => End Of File    NOF = No Operation  
MEM # => Memory Address Number in generator

Notice that a single quantization value has been used for all of the time intervals; this is not required in general, but provides a "nice" example. If several quantization values are used, the command column also contains information commanding the appropriate quantization clock for that specific interval. The time interval value represents the integer multiplier,  $a_k$ , shown above in the general equation. The number of pulse repetitions shown for each interval of a single pulse equals the same number, since there is an ON and an OFF time for a single pulse. The number of block repetitions is zero because there are no blocks in this sequence. If there were blocks of pulses, a block-start identifier is placed on the first time interval of the block, and an end-of-block identifier is placed on the last time interval of the block. A critical point is to notice that the identification of which

interval is the ON and which is the OFF is left out of the description. The assumption is that the first time interval counted represents (by default) the ON time, (i.e., the FW), and from this point the output simply toggles between OFF and ON forever. The definitions of a pulse and pulse sequence validate this assumption. If the first interval is an OFF time interval, the command column contains the information required to change the default ordering.

The last item required to realize the pulse sequence equation is some actualization of the random sequence. I.e. at the appropriate time, external values replace the data contained in the generator's memory, we now realize an aperiodic sequence. In fact, the data introduced into the system may originate from anywhere and be produced by any means. This modification allows tremendous flexibility in the generator. As configured in Figure 22, the pulse generator renders arbitrary pulse sequences of any length and configuration.

### 6.3 Realization of the generator

From Section 6.2 a block diagram of a pulse sequence generator is shown in Figure 22. Based on this diagram, a hardware device could now be created which accomplishes these functions. Although the actual hardware/software implementation of this device is beyond the scope of this thesis, I feel it is important to examine the possible

configurations for such a device.

#### 6.3.1 The MAPS system

A first-generation pulse generator exists at the Air Force Wright Aeronautical Laboratories, Avionics Laboratory, Electronic Warfare Division, AFWAL/AAWA-2, Wright-Patterson Air Force Base, Ohio. The system was christened the Multi-Agile Pulse Synthesizer due to its ability to render pulses of variable PW and PRI; and, in addition, the MAPS allows for variable length pulse sequences and pulse combinations. This first-generation system created pulses through the use of three 16-bit binary counter chains with one chain counting time intervals, one chain counting pulse repetitions, and the last chain counting block repetitions. Each chain was dedicated for use by only that division; therefore, unused counters remained dormant until called upon by the control hardware. This problem manifests itself in the situation where you need a 20-bit count capability but had only a 16-bit count available for that chain, even though only one or two bits were being used between the other two counter chains. Further, the control section did not support the pulse sequence description as developed in this thesis. Programming the MAPS required decoding the desired pulse sequence into a series of ones and zeroes, as discussed before. The user had to spend a great deal of time formatting the sequence into a usable form for the MAPS. The control section consisted of "fine tuned" hardwired

digital logic and was extremely difficult to maintain.

### 6.3.2 Other possible realizations

In this section I will examine some possible implementations of the generic synthesizer. The actual device depends on the available resources and skills at the time of design and construction. Since we now have an equation form of describing any arbitrary pulse sequence, the first division explored should be an input unit. Using the techniques of software engineering, the first step is to tokenize the input string via a parser. Given the input string as developed previously, the parser has little more to do than properly order an input command stack. This is because the sequence description is in token form already by using the symbols (, ), [, ], )+(, {, }, and the multipliers as specified in the notation. Since the generator must execute the description sequentially, little or no optimization is possible for the input. As described by Zelkowitz[19], if we stack the operands and execute the operators sequentially, we now have a direct execution machine. The control unit would simply pop the stack and supply that data to the processing elements.

As with the first-generation synthesizer, there is a need to count time intervals, pulse repetitions, and block repetitions. The pulse repetition increments only when the PW and PRI - PW time intervals have been counted. The block repetition count increments only when the proper number of

pulse repetitions of each pulse in the block has occurred. From these restrictions, a three-stage pipeline architecture for the processing elements (counter chains) is an appropriate configuration. Since each segment of the pipe accomplishes a different task for the same pulse, the pipe would be a processor pipeline. The pipeline performs a single function (unifunctional pipe) on scalar operands. The portion of the new generator performing the data processing is a three-stage processor pipeline operating in a unifunctional mode on scalar operands. If loading the pipeline requires a substantial amount of time, a possible solution would be to use an array processor configuration with each of the processing elements (PEs) being a single, three-stage pipe as described above. While one PE executes the data for a unique pulse, the other PEs could be loaded by the control unit and ready for execution upon completion of the current pulse. The interconnection network would simply be a demultiplexer output. This configuration allows for multiple sequence outputs when the control unit can load each PE fast enough.

The problem of unused counter bits was mentioned before in the first-generation synthesizer. Ideally, the PE would assign the appropriate number of counters to each pipe stage from a bank of counters located within that pipeline. The number of counters required depends on the magnitude of the multipliers used and their corresponding binary equivalent representation. Perhaps this job is best handled by the control unit; that is,

from the input data, the control unit decides how many counters are needed to count the time intervals, pulse repetitions, and block repetitions. From this decision, the control unit commands the appropriate PE to configure its counters accordingly. The interconnection network between the counters and the data input could be of several forms. The crossbar network allows any counter to be connected to the input as well as to other counters (for cascading); however, the cost of a crossbar network is of the order  $N^2$  where  $N$  is the number of interconnection input/output selectors in the PE. Certainly, a dynamic network is required; however, the cost of implementing a crossbar system may overshadow the cost of the system. Realistically, a counter will, at most, be connected to two other counters; therefore, any of the networks which are multistage in nature and allow up to two counter connections may be used. The crossbar and Clos networks fulfill these requirements as well as the Illiac-IV mesh connected network and some networks now being considered for arrays of processors.

#### 6.4 Chapter summary

A pulse sequence description is of no use unless it eases some algebra or can provide for an easy implementation of sequence generation. Since the notation described in Chapter 5 compactly describes a sequence in terms of easily implemented variables, the new notation seems very useful for the



application of describing and synthesizing an arbitrary sequence. A general model was discussed and some possible implementations were presented. The actual hardware device will be configured according to the budget allocations and available engineering expertise.

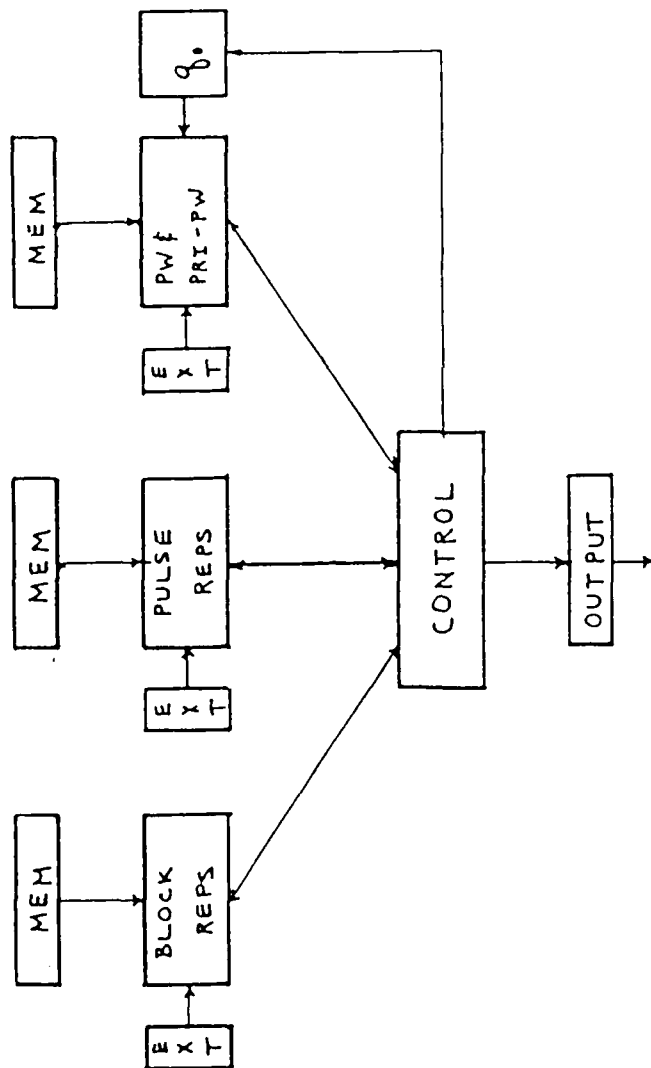


Figure 22 A block diagram of the generic pulse synthesizer.

## CHAPTER 7

### CONCLUSIONS

#### 7.0 Summary

The objective of this thesis was to find a means of describing an arbitrary pulse sequence, and should such a method be found, to try and implement the notation in some fashion. The Fourier transform is the engineer's standard tool to determine the spectral components of an input signal, so the use of the Fourier transform was examined in the first part of Chapter 3. The problem with this transform was that it yielded an infinite series which became unruly for even simple pulse sequences. Part of this is due to the Fourier transform attempting to identify all of the frequency components of a square wave. Since the Walsh transform produces a digital decomposition of the input, it was the next logical choice for a transform method. The Walsh function becomes messy and produces round-off errors when trying to transform an interval which is not an integer multiple of  $2^n$ . Also, both transforms required the designer to piecewise integrate the input digital sequence; this forced the user to identify every one and zero in the sequence.

Since transforms did not easily describe an arbitrary sequence, programming all of the ones and zeroes into a computer was tried next. In Chapter 4 the problems with computer execution were examined and it was found that large

amounts of memory are required for an arbitrary sequence of any length. Further, the software overhead required to step through memory produced significant limitations on the minimum value of time intervals that could be created through this means. A means of averting the speed limitation is to build a hardware device which generates the desired sequence through combinational logic. You gain the speed of a hardwired device but lose the programmability of a computer. For the computer and hardwired approaches the user still had to individually specify each one and zero in the sequence.

In Chapter 5 my shorthand notation for describing an arbitrary sequence was introduced. The symbology implies the necessary addition required to compute the sequence's period,  $T$ , and supplies the order of the ON/OFF sequences which describes the shape of the signal. An allowance is made for the instance where the sequence contains random information. After including this option, the new notation compactly and precisely describes an arbitrary pulse sequence.

A hardware/software implementation of the new notation is given in Chapter 6. A general block diagram explains the overall realization of the notation, and possible implementations were looked at next.

#### 7.1 Conjectures and future work

The utility of the new notation outside the realm of the pulse synthesizer is questionable. For long truth tables, the

notation provides a succinct description, but there is no algebra for the notation which could aid in reduction methods. Examination of an algebra for the notation is an area for possible study.

The area of acceptable error needs to be examined for the Walsh functions. When the interval is not an integer multiple of  $2^n$ , the Walsh functions produce  $MSE \neq 0$  only for the  $WAL(n,t)$  where  $n$  equals infinity. Depending on the acceptable error, the Walsh functions may prove to be useful with the proper error algorithm.

## BIBLIOGRAPHY

- [1] D. Smith, Digital Transmission Systems.  
New York: Van Nostrand Reinhold Company, 1985, p. 4.
- [2] J. Dart, D. Burum, W. Rhim, "Highly flexible pulse  
programmer for NMR applications,"  
Review of Scientific Instruments, vol. 51, no.2,  
February 1980, pp. 224-228.
- [3] M. Skolnik, Introduction to Radar Systems.  
New York: McGraw-Hill Book Company, 1980, p. 2.
- [4] D. Johnson, J. Hilburn, J. Johnson,  
Basic Electric Circuit Analysis. New Jersey:  
Prentice-Hall, 1978, pp. 182-186.
- [5] R. Wiley, Electronic Intelligence: the Analysis of  
Radar Signals. Dedham, Massachusetts:  
Artech House, Inc., 1982, pp. 127-135.
- [6] W. Chambers, Basics of Communications and Coding.  
Oxford: Clarendon Press, 1985, pp. 13-38.
- [7] S. Shinnars, Modern Control System Theory and  
Application. Massachusetts: Addison-Wesley  
Publishing Co., 1978, pp. 27-34.
- [8] R. Beauchamp, Applications of Walsh and Related  
Functions with an Introduction to Sequence Theory.  
London: Academic Press, 1984.

- [99] J. Beauchamp, Applications of Walsh and Related Functions with an Introduction to Sequence Theory.  
London: Academic Press, 1984, pp. 18-23.
- [100] J. Beauchamp, Applications of Walsh and Related Functions with an Introduction to Sequence Theory.  
London: Academic Press, 1984, pp. 24-31.
- [111] C. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and two-level logic synthesis," IEEE Trans. Comp., C-24, no. 1, January 1975, pp. 48-62.
- [112] L. Bennett, "Realisation of logic functions by multi-output threshold-logic gates," IEE Proc., vol. 129, pt. E, no. 5, November 1982, pp. 232-243.
- [113] P. Picton, "Realisation of multithreshold threshold logic networks using the Rademacher-Walsh transform," IEE Proc., vol. 129, pt. E, no. 3, May 1981, pp. 107-113.
- [114] D. Haring, D. Onori, "A tabular method for the synthesis of multi-threshold threshold elements," IEEE Trans. Comp., EC-16, April 1967, pp. 215-220.
- [115] S. Golomb, Shift Register Sequences.  
San Francisco: Holden-Day, Inc., 1967.
- [116] S. Golomb, Shift Register Sequences.  
San Francisco: Holden-Day, Inc., 1967, p. 142.
- [117] S. Golomb, Shift Register Sequences.  
San Francisco: Holden-Day, Inc., 1967, p. 28.

ATE  
LMED  
=8